Documentation of ModelSolver
A Python class for analyzing dynamic algebraic models

Magnus Kvåle Helliesen

March 23, 2023

Abstract

This paper documents the Python class ModelSolver. The class lets the user define a model object in terms of equations and endogenous variables. It contains methods to solve the model subject to data in a Pandas DataFrame, as well as analyzing the model.

ModelSolver was developed in order to facilitate solving a monthly input-output model for the Norwegian monthly national accounts.

1 Background

1.1 The model

Suppose we have a model, [1]

$$
\begin{aligned}
L_1(\mathbf{x}_t, \mathbf{z}_t) &= H_1(\mathbf{x}_t, \mathbf{z}_t),\\
L_2(\mathbf{x}_t, \mathbf{z}_t) &= H_2(\mathbf{x}_t, \mathbf{z}_t),\\
&\vdots\\
L_n(\mathbf{x}_t, \mathbf{z}_t) &= H_n(\mathbf{x}_t, \mathbf{z}_t),
\end{aligned}
$$

for $t = T_0, T_0 + 1, \ldots, T_1$, where $\mathbf{x}_t = (x_{1,t}, x_{2,t}, \ldots, x_{n,t})$ is a vector of endogenous variables, and $\mathbf{z}_t = (z_{1,t-k_1}, z_{2,t-k_2}, \ldots, z_{m,t-k_m}, \ldots, x_{i,t-\kappa_i}, \ldots)$, for $k_i \geq 0$ and $\kappa_i > 0$, is a vector of simultaneous and lags of exogenous variables and lags of endogenous variables. [2] It's convenient to formulate the model on vector form,

$$
\mathbf{F}(\mathbf{x}_t, \mathbf{z}_t) = \begin{pmatrix} F_1(\mathbf{x}_t, \mathbf{z}_t) \\ F_2(\mathbf{x}_t, \mathbf{z}_t) \\ \vdots \\ F_n(\mathbf{x}_t, \mathbf{z}_t) \end{pmatrix} = \begin{pmatrix} L_1(\mathbf{x}_t, \mathbf{z}_t) - H_1(\mathbf{x}_t, \mathbf{z}_t) \\ L_2(\mathbf{x}_t, \mathbf{z}_t) - H_2(\mathbf{x}_t, \mathbf{z}_t) \\ \vdots \\ L_n(\mathbf{x}_t, \mathbf{z}_t) - H_n(\mathbf{x}_t, \mathbf{z}_t) \end{pmatrix},
$$

https://github.com/statisticsnorway/model-solver.git
mkh@ssb.no

[1] L_i and H_i denote the left and right hand side of equation i.

[2] Note that $\mathbf{z}_t$ may contain only $z_{i,t-k_i}$'s or $x_{i,t-k_i}$'s or be empty.

Figure 1: Bipartite graph (BiGraph of model)

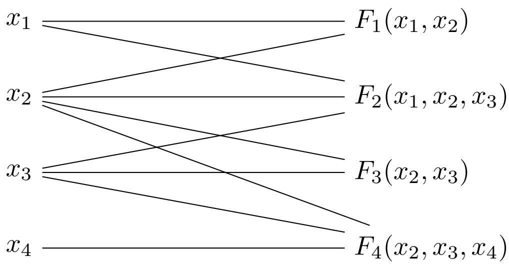
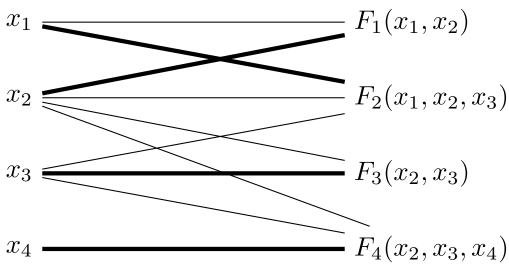


Figure 2: Maximum Bipartite Match (MBM) of BiGraph



with the solution to the model being given by $\{\mathbf{x}_t\}_{t=T_0}^{T_1}$ such that

$$\mathbf{F}(\mathbf{x}_t, \mathbf{z}_t) = \mathbf{0} \text{ for } t = T_0, T_0 + 1, \ldots, T_1.$$

On the face of it, this is a problem which the *Newton-Raphson*-algorithm handles well. The issue, however, is that n might be quite large. In the Norwegian national accounts, for instance, n is more than 15,500. Therefore, it is useful to analyze the system of equations before solving it, in order to break it down into minimal simultaneous blocks that can be solved in a particular sequence.

1.2 Block analysis

In order to analyze and divide the model into blocks, we use results from *graph theory*.

First, we construct a *bipartite graph* (BiGraph) that connects endogenous variables with equations. This is illustrated in Figure 1 for an arbitrary model with 4 equations (we omit time subscripts and exogenous variables and lags for notational convenience).

Next, we apply *maximum bipartite matching* (MBM), which assigns each endogenous variable to *one and only one* equation. The MBM is arbitrary, and Figure 2 shows one possible solution.

We use Figure 2 to determine what endogenous variables impact what *other* endogenous variables. This is illustrated in Figure 3.

Next, we use Figure 3 to construct a *directed graph* (DiGraph) that shows how the endogenous variables impact each other. The DiGraph is shown in Figure 4.

Figure 3: Graph of what endogenous variables impact what *other* endogenous variables

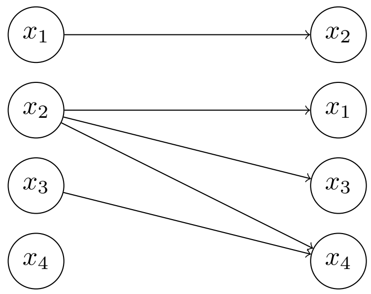


Figure 4: Directed graph (DiGraph) of what endogenous variables impact what *other* endogenous variables

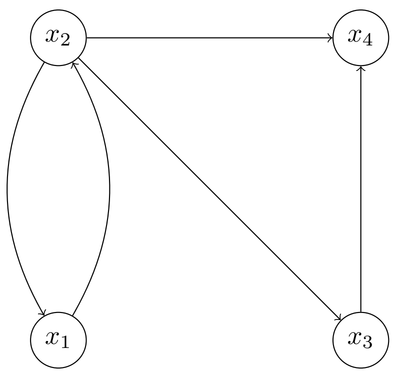


Figure 5: Condensation of DiGraph

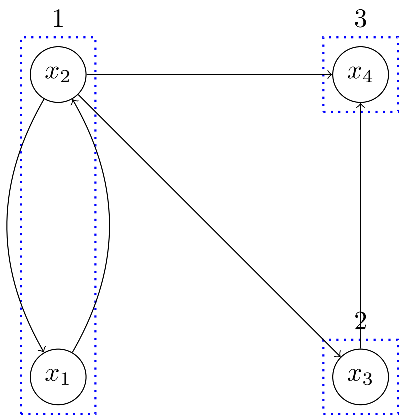
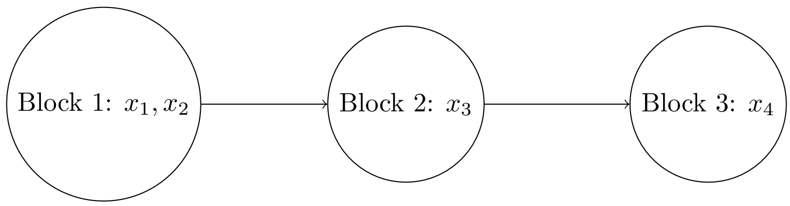


Figure 6: Condensed DiGraph



Finally, we find the *strong components* of the DiGraph, as shown in Figure 5. A strong component is a set of nodes that are connected such that every node can be reached from every other node in the set (traversing the arrows). A *condensation* of the DiGraph is a (new) DiGraph with each node being a strong component (of the former). The condensation can be illustrated as in Figure 6. Each node of the condensation corresponds to a block of the model, and the arrows decide the sequence in which the blocks are to be solved. See [1] for a discussion of strong components and block analysis.

In this example, x_1 and x_2 must be solved first in one simultaneous block. Next, x_3 is solved (taking x_1 and x_2 as given by the solution to block 1). Finally, x_4 is solved (taking x_1, x_2 and x_3 as given by the solutions to block 1 and 2).

1.3 Simulation code

With the blocks of the model as given by the condensation of the model DiGraph as previously discussed, we can generate *simulation code* for each block. That is *either*

> a function (definition) that takes exogenous input and returns the endogenous value (if the block is a definition) as described below, *or*

> a symbolic *objective function* and *Jacobian matrix* that takes exogenous input and (initial) values for the endogenous variables, which are to be sent to a Newton-Raphson algorithm.

A block is a said to be a *definition if and only if* it 1) consists of one equation and 2) the only thing on the left hand side of the equation is the endogenous variable of that equation (and that endogenous variable does *not* show up on the right hand side of that equation too).

1.4 Solution

In order to solve the model numerically, we loop over time periods (chronologically) and blocks (according to the order dictated by the condensation of the model DiGraph).

If the block is a definition, as discussed above, the solution is given by the function defining it (given exogenous and predetermined variables).

If the block to be solved is not a definition, it is sent to a Newton-Raphson algorithm, which in turn returns the solution (given exogenous and predetermined variables, and initial values for the endogenous variables). The $k+1$st iteration of the Newton-Raphson algorithm is given by

$$\mathbf{x}_t^{(k+1)} = \mathbf{x}_t^{(k)} - \mathbf{J}_{\mathbf{F}}^{-1}(\mathbf{x}_t^{(k)}, \mathbf{z}_t)\mathbf{F}(\mathbf{x}_t^{(k)}, \mathbf{z}_t),$$

where $\mathbf{J}_{\mathbf{F}}(\mathbf{x}_t, \mathbf{z}_t)$ is the Jacobian matrix of $\mathbf{F}(\mathbf{x}_t, \mathbf{z}_t)$ evaluated at $\mathbf{x}_t$ and $\mathbf{z}_t$. The algorithm stops and returns $\mathbf{x}_t^{(k+1)}$ if $\max\left(\left|\mathbf{x}_t^{(k+1)} - \mathbf{x}_t^{(k)}\right|\right) \leq \varepsilon$, where ε is a tolerance level. If $\mathbf{F}(\mathbf{x}_t^{(0)}, \mathbf{z}_t) = \mathbf{0}$ then the initial values solve the model, and as such $\mathbf{x}_t^{(0)}$ is returned as the solution.

2 Examples of use

The `ModelSolver` object is instantiated by

```
model = ModelSolver(equations, endogenous)
```

where `equations` and `endogenous` are lists containing equations and endogenous variables as strings in lists, e.g.,

```
equations = [
'x1 = a1',
'x2 = a2',
'0.2*x1+0.7*x2 = 0.1*ca+0.8*cb+0.3*i1',
'0.8*x1+0.3*x2 = 0.9*ca+0.2*cb+0.1*i2',
'k1 = k1(-1)+i1',
'k2 = k2(-1)+i2'
]
endogenous = ['x1', 'x2', 'ca', 'cb', 'k1', 'k2']
```

Note that `(-#)` signifies lags of a variable. `ModelSolver` does not support leads. Mathematical functions supported are `max()`, `min()`, `log()` and `exp()`.

It is possible to switch endogenous and exogenous variables using

```
model.switch_endo_vars(old_endo_vars, new_endo_vars)
```

where `old_endo_vars` and `new_endo_vars` are lists.

2.1 Solving the model

Let `input_df` be a `Pandas DataFrame` containing exogenous variables and initial values for the endogenous variables. Then

```
solution_df = model.solve(input_df)
```

solves the model and stores the results in `solution_df`, which is a `Pandas DataFrame` identical to `input_df`, *but* where the endogenous variables are solutions to the model.

2.2 Analyzing the model

`ModelSolver` contains a number of methods that lets the user analyze the model.

`describe()` describes the model in terms of number of equations and blocks, along with information about the blocks, e.g.

```
Model consists of 15568 equations in 14358 blocks
14159 of the blocks consist of simple definitions

14348 blocks have 1 equations
4 blocks have 2 equations
1 blocks have 3 equations
2 blocks have 4 equations
1 blocks have 7 equations
1 blocks have 132 equations
1 blocks have 1062 equations
```

`find_endo_var(<endogenous variable>)` returns the block in which the endogenous variable is solved for.

`show_block(<block>)` returns information about the block, i.e. endogenous variables, exogenous and predetermined variables and equations.

`show_blocks()` returns information about all the blocks of the model.

`trace_to_exog_vars(<block>)` traces the model DiGraph back to the nodes of origin and reports what exogenous variables determine those nodes.

`trace_to_exog_vals(<block>, <period index>)` traces the model DiGraph back to the nodes of origin, finds what exogenous variables determine those nodes and reports values for those variables for the chosen period index (0,1,...).

`draw_blockwise_graph()` draws a network graph as shown in Figure 7. The method takes as arguments a variable name, and the maximum number of ancestor and descendant nodes, and maximum number of nodes in total.

References

[1] Gilli, Manfred. "Causal Ordering and Beyond." International Economic Review, vol. 33, no. 4, 1992, pp. 957–71. JSTOR, https://doi.org/10.2307/2527152. Accessed 22 Mar. 2023.

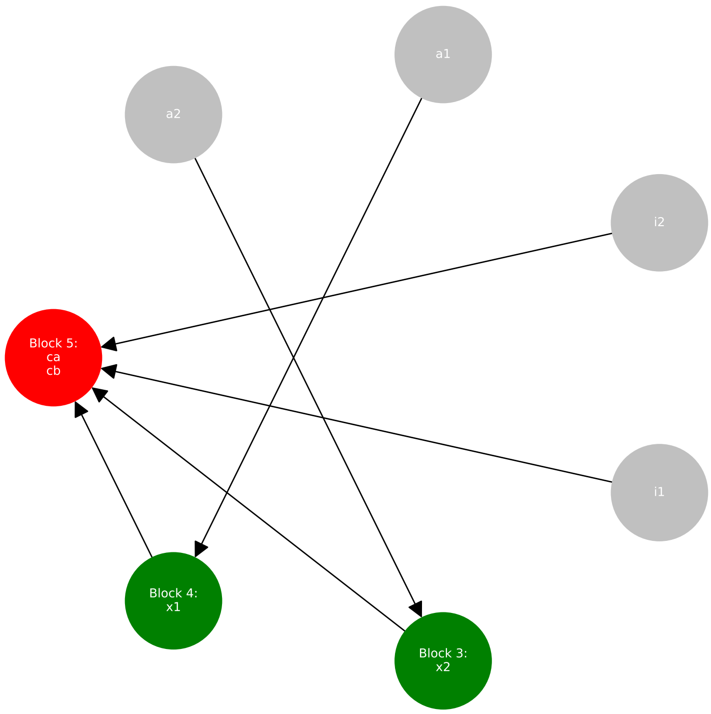


Figure 7: Example of network plot